Precis

Poster

Agenda

# Title: Pulse Compression Made Easy[1] With VSIPL++

Authors

First Author: **Mr. Brian Chase**
(US citizen)
VSI/Pro Product Manager
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 209
Fax: +1 (205) 314-3475
E-mail: brian@mpi-softtech.com

Second Author: **Mr. WenhaoWu**
(Citizen of China)
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 209
Fax: +1 (205) 314-3475
E-mail: wenhao@mpi-softtech.com

Third Author: **Mr. Dave Leimbach**
(US Citizen)
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 209
Fax: +1 (205) 314-3475
E-mail: dleimbac@mpi-softtech.com

Fourth Author: **Mr. Rick Pancoast**
(US Citizen)
Lockheed Martin Naval Electronics and Surveillance Systems - Surface Systems
199 Borton Landing Road
Moorestown, NJ  08057
E-mail: rick.pancoast@lmco.com

Corresponding and Presenting Author: **Dr. Anthony Skjellum**
(US citizen)
Chief Software Architect,
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 205
Fax: +1 (205) 314-3475
E-mail: tony@mpi-softtech.com

Submission session: Open sessions.
Presentation type: Presentation

Work area: Case Study Examples of High Performance Embedded Computing

[1] High Productivity – See the abstract for more details.

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| 1. REPORT DATE<br>**01 FEB 2005** | 2. REPORT TYPE<br>**N/A** | 3. DATES COVERED<br>**-** |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>**Pulse Compression Made Easy With VSIPL++** | 5a. CONTRACT NUMBER |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**Verari Systems Software, Inc. Suite D103, 110 12th Street North Birmingham, AL 35203** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT

**Approved for public release, distribution unlimited**

13. SUPPLEMENTARY NOTES

**See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **UU** | **13** | |

In December, 2003, Verari Systems Software, Inc. (formerly MPI Software Technology, Inc.) undertook a phase I SBIR effort produce a high level design for a high performance next generation embedded VSIPL product that incorporates advanced language constructs such as those found in the VSIPL++ specification that is now under consideration. The work was divided as follows: 1) Researching strategies to mitigate performance degradation from C++ overhead; 2) High level design work for such a library; 3) Prototype implementation of the new library; 4) Implementing a benchmark application to gauge performance benefit; 5) Reporting results and making recommendations that would apply to the ongoing VSIPL++ effort and any follow-on Phase II work that might be awarded.

The recent introduction of several template based strategies (*e.g.* PETE, POOMA, FACT!, Blitz++, and others ) suggests that C++ may soon become a suitable choice for technical and scientific computing application. For certain cases, (*e.g.*, matrix multiplications) inline function calls and template code outperforms straight C code. These similar technologies all share a common set of effective strategies that may be summarized as follows: 1) Avoid excessive temporary copies of objects (both implicit and explicit, where the implicit ones are generated as a side effect of algebraic type expressions); 2) Make shallow copies instead of deep cloning; 3) Pass data by constant reference instead of by value; 4) Use compile time or static polymorphism, such as templates; 5) Deferred evaluation; 6) Template metaprogramming strategies; 7) Inline function calls; 8) Loop fusion and loop unrolling. The authors acquired PETE, the Portable Expression Template Engine, and compiled several examples for the Mercury MCOE 6.0 platform using a 171MHz SPARC machine. The authors also studied the tradeoffs between runtime performance and significant compile time penalties.

Verari’s advanced VSIPL package design and prototype implementation strategy is not unlike the architecture of the VSIPL++ reference implementation, which is built as a C++ layer on top of a C VSIPL library. That particular configuration readily appeals to all current vendors of VSIPL compliant middleware who would like to quickly enter the market with a VSIPL++ offering. Figure 1 shown below depicts the layered hierarchical software design used in this study.

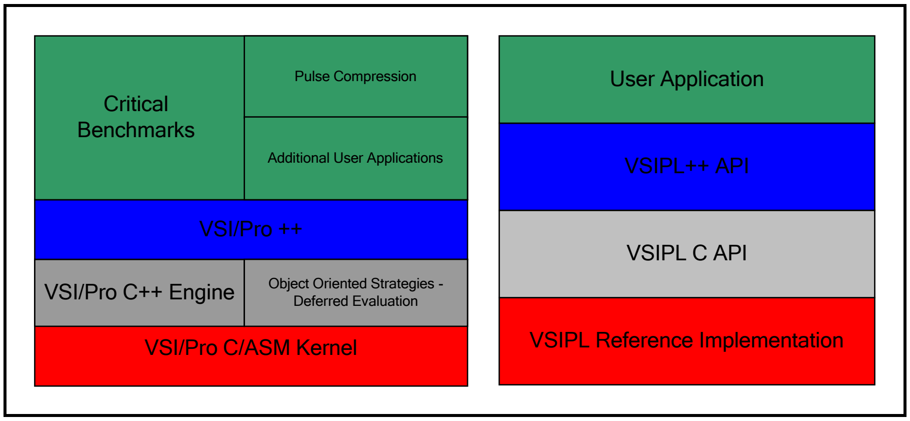


**Figure 1 Architecture or the prototype versus VSIPL++**

Since the API of the prototype package mirrors that of the VSIPL++ reference implementation, the first experiments were simply tests that are distributed with the VSIPL++ reference code. These tests mainly check for numerical accuracy. The Phase I study then progressed to a more complex test, a commonplace benchmarking application used in radar processing. The pulse compression benchmark typically uses a complex FFT, a complex reference multiply, followed by an inverse complex FFT. The performance of the prototype library on this benchmark was inline with performance figures than can be obtained from the VSI/Pro package. Since the VSI/Pro++ API is similar to VSIPL++, we should expect high performance from the VSIPL++ API when the time comes.

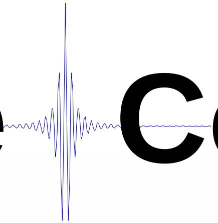
Other than improved performance, another noticeable observation that occurred while porting the pulse compression application from VSIPL to VSIPL++ was the dramatic reduction in both code size and complexity: The original VSIPL benchmark code (which was provided by Lockheed Martin as part of this SBIR effort) consisted of 1600 lines of C code. Yet, the ported VSIPL++ code consisted only 100 lines of C++ code and the whole porting effort only required 2 weeks for one engineer. In fact, Pulse Compression can be fully implemented in a single line of VSIPL++ code:

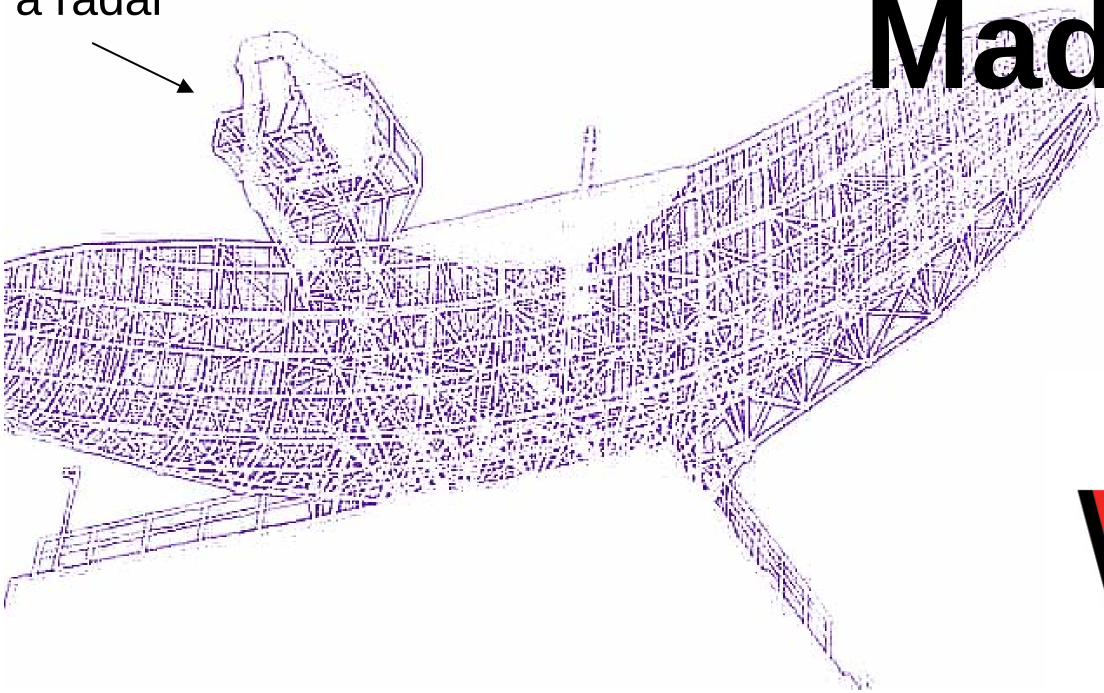
```
OutputVector = fft_ccrv (fft_ccfv (InputVector) * fft_ccfv (WeightVector));
```

In conclusion, this layered software architectural approach enables high performance, portability, high productivity, and low time to market for commercial vendors of VSIPL standard libraries. Future directions include incorporating the following strategies that will facilitate commercialization of the VSIPL++ standard: 1) Generic programming for higher productivity. 2) Expression manipulation, as well as 3) Deferred evaluation for higher performance.

Poster
Agenda

Verari systems software™

# Pulse  Compression Made Easy with VSIPL++

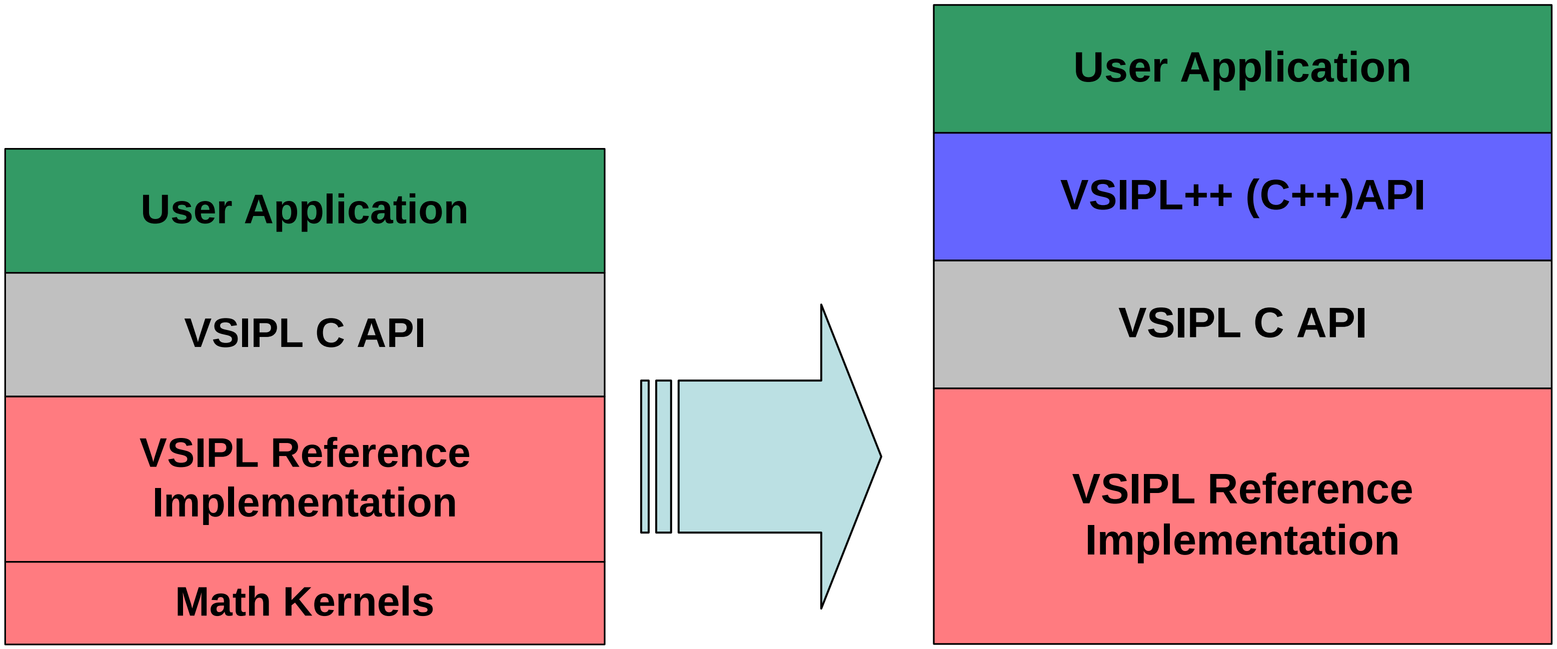


Verari systems software™

Funded Under SBIR
Topic OSD03-022 (OSD/AF)
“High Performance Object Oriented Software for
Parallel Embedded Systems”

Formerly MPI Software Technology, Inc.

# VSIPL and VSIPL++ Reference Implementations



User Application

VSIPL C API

VSIPL Reference Implementation

Math Kernels

User Application

VSIPL++ (C++)API

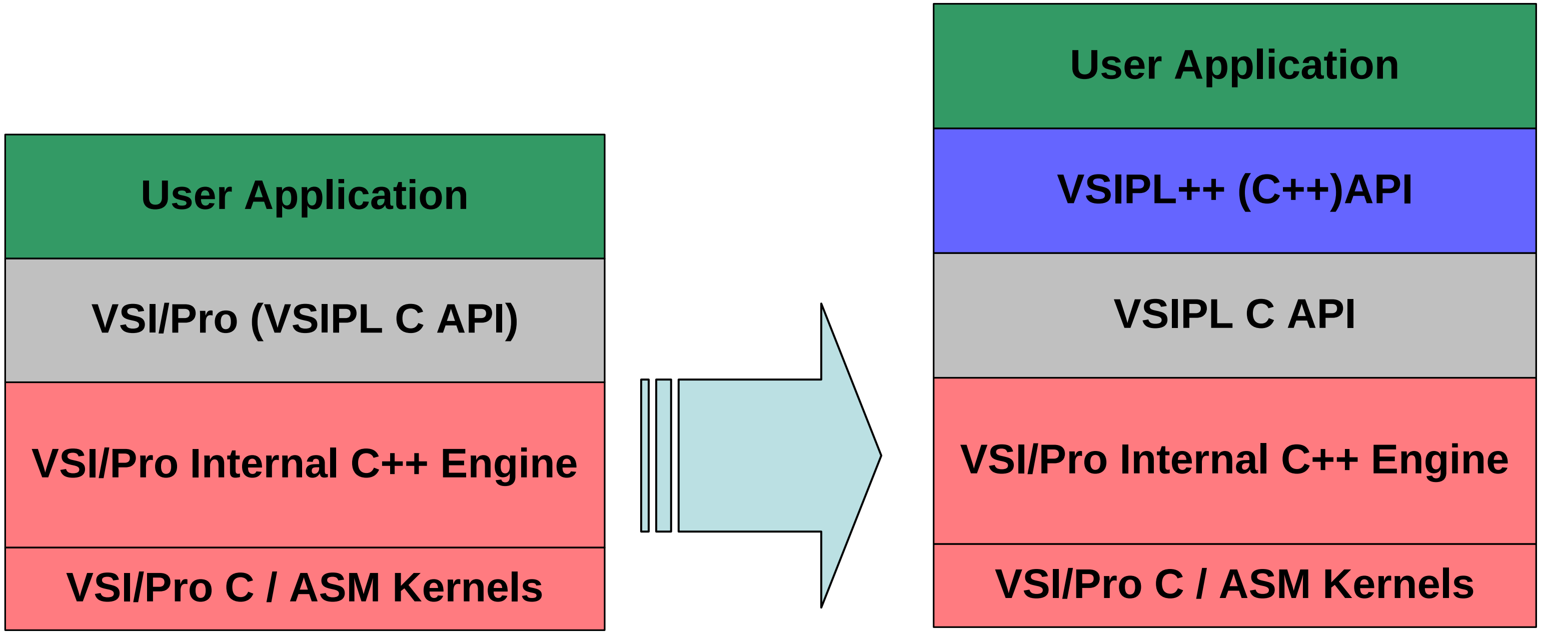
VSIPL C API

VSIPL Reference Implementation

**The VSIPL Reference Implementation**

**The VSIPL++ Reference Implementation**
**Builds upon the VSIPL Reference Implementation**

# VSI/Pro Product and the VSI/Pro++ Prototype



User Application

VSI/Pro (VSIPL C API)

VSI/Pro Internal C++ Engine

VSI/Pro C / ASM Kernels

User Application

VSIPL++ (C++)API

VSIPL C API

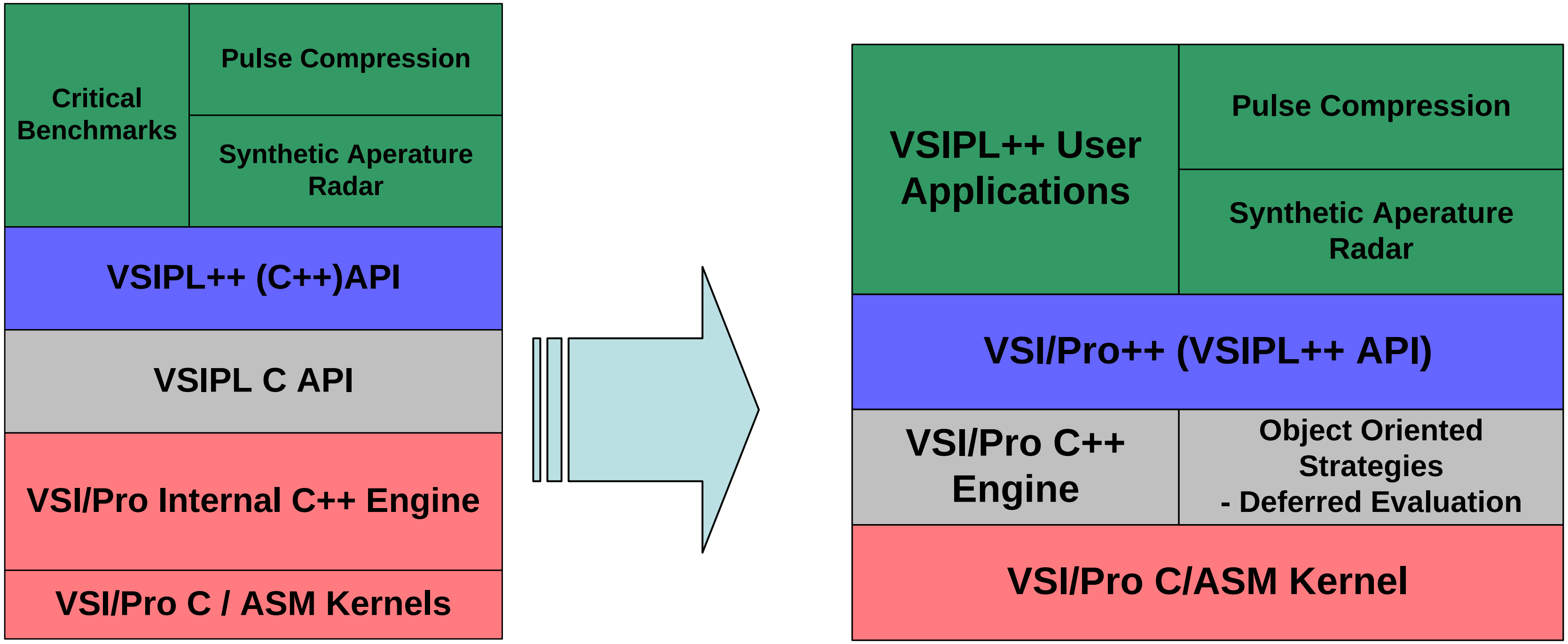
VSI/Pro Internal C++ Engine

VSI/Pro C / ASM Kernels

Structure of VSI/Pro

The VSI/Pro++ Prototype
Builds upon the VSI/Pro Product

# Layered Approach versus a Pure Implementation



Critical Benchmarks

Pulse Compression

Synthetic Aperature Radar

VSIPL++ (C++)API

VSIPL C API

VSI/Pro Internal C++ Engine

VSI/Pro C / ASM Kernels

VSIPL++ User Applications

Pulse Compression

Synthetic Aperature Radar

VSI/Pro++ (VSIPL++ API)

VSI/Pro C++ Engine

Object Oriented Strategies
- Deferred Evaluation

VSI/Pro C/ASM Kernel

- What are the benefits of a Pure VSI/Pro++ Product.
- Having both API bindings available is a hidden benefit to programs that  want to migrate their systems from VSIPL to VSIPL++ in phases.

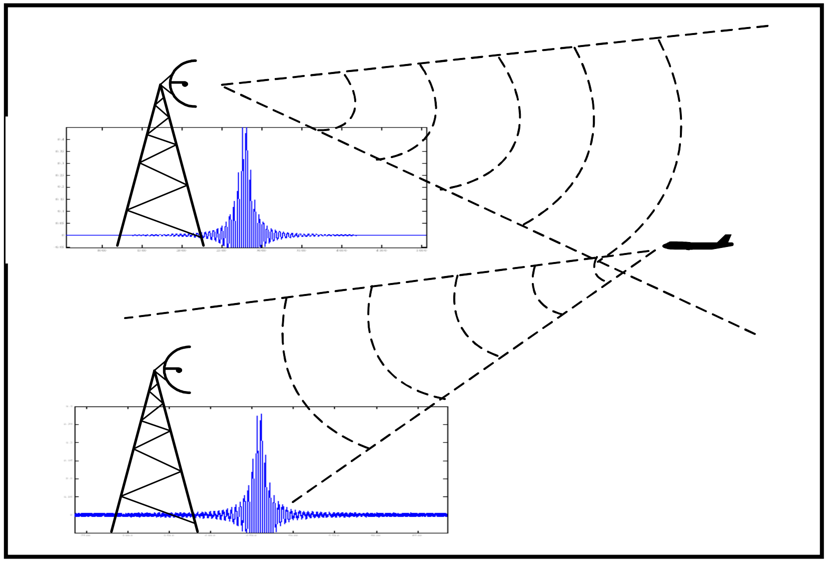
# Performance Comparison for 1024 Point Complex FFT

| | Data Size | CCFFT by value | CCFFT by reference |
|---|---|---|---|
| VSIPL (VSI/Pro) | 1024 | does not apply | 11.52 us |
| VSIPL++ (VSI/Pro) | 1024 | 18.74 us | 12.24 us |
| | | | |
| | | multiple CCFFT by value | multiple CCFFT by reference |
| VSIPL (VSI/Pro) | 1024 sets of 1024 | does not apply | 80 ms |
| VSIPL++ (VSI/Pro) | 1024 sets of 1024 | 127.540 ms | 82.350 ms |

- Did not experience any significant overhead from layering the VSIPL++ API on top of the VSI/Pro API (See rightmost column).

# Case Study: Pulse Compression

Pulse Compression works by distributing the energy in the outgoing Radar pulse over a larger span of time with one of a select number of waveform pulses that are generally known as chirp waveforms. This kind of filtering not only improves the accuracy of the measurements, but also rejects most kinds of ambient noise. The net effect is an improvement in resolution and decreased demand for peak power requirement in the signal generation equipment. A typical pulse consists of a short burst of frequency like the one shown here.



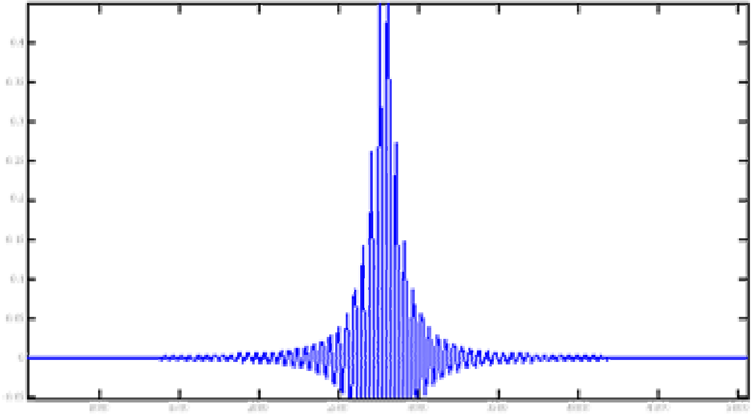
The digital signal processing functions that are associated with pulse compression applications typically use a complex FFT, a complex reference multiply, followed by an inverse complex FFT. Pulse compression, and FFT processing in general comprise a major portion of the processing load in state of the art radar systems.

# Pulse Compression: The VSIPL way

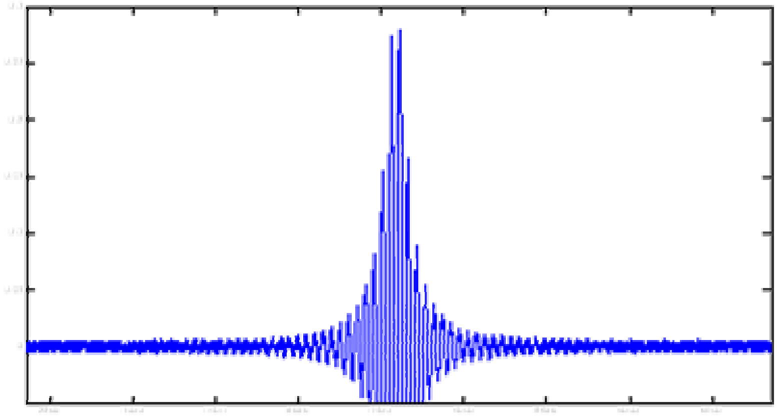
```
The pseudocode:
Create Vectors
Create Forward FFT object
Create Inverse FFT object

Create 3 temporary vector to hold intermediate frequency domain results.

Convert reference signal vector to the frequency domain:
Forward FFT( Ref Signal Vec, Temp Vec1 )

Convert Input signal to the frequency domain:
Forward FFT( Input Signal Vec, Temp Vec2 )

Multiply vectors in the frequency domain:
Vector Multiply( Temp Vec1, Temp Vec2, Temp Vec3 )

Obtain the inverse FFT:
Inverse FFT( Temp Vec3, Answer Vec )
```

# Pulse Compression: The VSIPL++ way

```
The pseudocode:
Create Vectors
Create Forward FFT object
Create Inverse FFT object

Answer Vec = INV_FFT( FFT(Input Vec)*FFT(Reference Signal Vec));
```



Reference Signal



Noisy Return

# Integrating Expression Manipulation into VSIPL++

Expression object strategies address the important problem of temporary copy proliferation that occurs as a result of operator overloading in C++.

Existing technologies that were studied -

- PETE (Portable Expression Template Engine)

  Developed at the Advanced Computing Laboratory at the Los Alamos National Laboratory
- BLITZ++

  The goal of Blitz++ is to provide a similar level of performance on par with Fortran 77/90
- FACT! (Functional Additions to C++ through Templates and Classes )

  A library that provides expression manipulation plus other functional programming language features not normally accessible in C++.

# Observations from using VSIPL++

Benefits:

- Concise code
- Readable
- Natural looking expressions

Hazards:

- Complex looking data types, may be helped in practice by typedefs
- General C++ concerns (e.g., possible to abuse the language)